

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2002-024029
 (43)Date of publication of application : 25.01.2002

(51)Int.CI. G06F 9/45

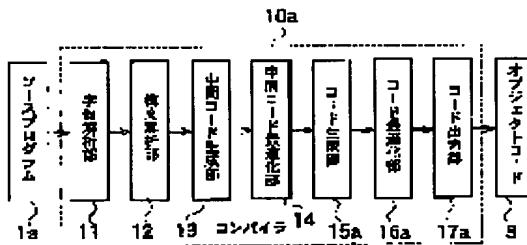
(21)Application number : 2000-210412 (71)Applicant : TOSHIBA CORP
 (22)Date of filing : 11.07.2000 (72)Inventor : HAYASHIDA SEIJI

(54) COMPILER, COMPIILING METHOD AND COMPUTER READABLE RECORDING MEDIUM WITH COMPILE PROGRAM RECORDED THEREON

(57)Abstract:

PROBLEM TO BE SOLVED: To easily expand a compiler by making a built-in function (intrinsic function) definable when or while a compiler operates.

SOLUTION: This compiler 10 for generating an object code 3 with a source program 1 as an input has a lexical analyzing part 11 of dividing an instruction described in the program 1 into tokens, a syntax analyzing part 12 for analyzing whether the tokens are suitable for grammar, a code generating part 15 for converting the program 1 into a machine language, and a built-in function definition attribute storing means for storing a built-in function definition attribute. The syntax analyzing part 12 analyzes the definition and attributes of the built-in function and stores the definition and attributes in the built-in function definition attribute storing means.



LEGAL STATUS

[Date of request for examination] 28.01.2003

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

Copyright (C); 1998,2003 Japan Patent Office

(19)日本国特許庁 (JP)

(12) 公開特許公報 (A)

(11)特許出願公開番号

特開2002-24029

(P2002-24029A)

(43)公開日 平成14年1月25日 (2002.1.25)

(51)Int.Cl.⁷

G 0 6 F 9/45

識別記号

F I

G 0 6 F 9/44

テ-マコ-ト^{*}(参考)

3 2 2 A 5 B 0 8 1

審査請求 未請求 請求項の数9 O L (全 8 頁)

(21)出願番号 特願2000-210412(P2000-210412)

(22)出願日 平成12年7月11日 (2000.7.11)

(71)出願人 000003078

株式会社東芝

東京都港区芝浦一丁目1番1号

(72)発明者 林田 聖司

神奈川県川崎市幸区小向東芝町1番地 株式会社東芝マイクロエレクトロニクスセンター内

(74)代理人 100083806

弁理士 三好 秀和 (外7名)

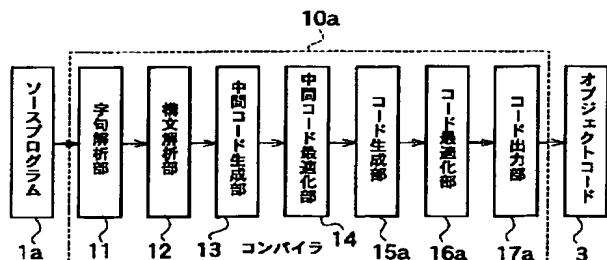
Fターム(参考) 5B081 AA06 AA07 CC41

(54)【発明の名称】 コンパイラ、コンパイル方法及びコンパイルプログラムを記録したコンピュータ読み取り可能な記録媒体

(57)【要約】

【課題】 組込関数 (intrinsics関数) をコンパイラの起動時又は動作中に定義可能として、コンパイラの拡張を容易にする。

【解決手段】 ソースプログラム1を入力しオブジェクトコード3を生成するコンパイラ10であって、ソースプログラム1内に記述された命令をトークンに分割する字句解析部11と、トークンが文法に適しているかを解析する構文解析部12と、ソースプログラム1を機械語へ変換するコード生成部15と、組込関数定義属性を記憶する組込関数定義属性記憶手段を有し、構文解析部12が、組込関数の定義と属性を解析し、組込関数定義属性記憶手段に記憶する。



【特許請求の範囲】

【請求項1】 ソースプログラム中あるいはヘッダファイル中の組込み関数の定義及び属性を読み取り、前記組込み関数の定義及び属性に基づいて前記ソースプログラムを機械語に変換することを特徴とするコンパイラ。

【請求項2】 ソースプログラムを入力としオブジェクトコードを生成するコンパイラであって、

前記ソースプログラム内に記述された命令をトークンに分割する字句解析部と、

前記トークンが文法に適しているかを解析し、これらトークンの組み合わせが組込み関数及び属性を定義しているかを解析する構文解析部と、

前記構文解析部で解析された組込み関数の定義及び属性を記憶する組込み関数定義属性記憶手段と、

前記組込み関数の定義及び属性に基づいて、前記ソースプログラムを機械語へ変換するコード生成部とを有することを特徴とするコンパイラ。

【請求項3】 前記組込み関数の定義に、仮引数型と識別指名が含まれることを特徴とする請求項1又は2記載のコンパイラ。

【請求項4】 コンパイラの起動時または動作中に、組込み関数の定義が開始する旨の宣言を検出する工程と、前記組込み関数の定義を登録する工程と、

前記組込み関数の属性に関する記述が開始する旨の宣言を検出する工程と、

前記組込み関数の属性を登録する工程とを少なくとも含むことを特徴とするコンパイル方法。

【請求項5】 コンパイラの起動時または動作中に、組込み関数の定義及び組込み関数の属性に関する記述が開始する旨の宣言を検出する工程と、

前記組込み関数の定義を登録する工程と、

前記組込み関数の属性を登録する工程とを少なくとも含むことを特徴とするコンパイル方法。

【請求項6】 前記組込み関数の定義に、仮引数の型、識別子名が含まれることを特徴とする請求項4又は5記載のコンパイル方法。

【請求項7】 コンパイラの起動時または動作中に、組込み関数の定義が開始する旨の宣言を検出するステップと、

前記組込み関数の定義を登録するステップと、

前記組込み関数の属性に関する記述が開始する旨の宣言を検出するステップと、

前記組込み関数の属性を登録するステップとを少なくとも含むことを特徴とするコンパイルプログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項8】 コンパイラの起動時または動作中に、組込み関数の定義及び組込み関数の属性に関する記述が開始する旨の宣言を検出するステップと、

前記組込み関数の定義を登録するステップと、

前記組込み関数の属性を登録するステップとを少なくと

も含むことを特徴とするコンパイルプログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項9】 前記組込み関数の定義に、仮引数の型、識別子名が含まれることを特徴とする請求項7又は8記載のコンパイルプログラムを記録したコンピュータ読み取り可能な記録媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】 本発明は、C言語などの高級言語を機械語に変換（コンパイル）するコンパイラ、コンパイル方法及びコンパイルプログラムを記録したコンピュータ読み取り可能な記録媒体に関し、特に組込み関数（intrinsics関数）をコンパイラの起動時又は動作中に任意に追加定義可能とする技術に関する。

【0002】

【従来の技術】 従来、例えば、mov命令（mov Rn, Rm : レジスタRmの内容をレジスタRnに代入する）に対応する回路構成をプロセッサが有している場合でも、コンパイラがmov命令に対応していない場合には、ユーザであるプログラム作成者はmov命令を使った高級言語によるプログラムを作成することができない。このような場合、プログラム作成者は使用するプロセッサの構造がmov命令に対応しているにも関わらず、"mov Rn, Rm"と記述する代わりに、同機能を表すためにはlw(load word), sw(store word)命令を用いてプログラムを記述しなければならない。すなわち、プロセッサはmov命令に対応する回路構成を持ちながら、lw, swなどの複数の命令を実行しなければならない。

【0003】 また、单一命令で複数データを並列処理するマルチメディア命令のように、既存命令の組み合わせでは実現できないような命令に対応していないコンパイラを使用した場合には、プログラム作成者はプロセッサ特有の機能を利用することができない。

【0004】 従って、プロセッサ提供側は、ユーザがプロセッサを効率よく動作可能とするために、プロセッサ特有の仕様（プロセッサ・アーキテクチャおよび命令セット）に対応したコンパイラを提供しなければならない。

【0005】 プロセッサ特有の仕様（プロセッサ・アーキテクチャおよび命令セット）にコンパイラを対応させる手法としては、intrinsics関数（プログラム言語の処理系が予め用意している関数であり、対象となるプロセッサ特有の高級言語による命令記述を機械語にコンパイルさせるための組込み関数）を用いる手法が存在する。

【0006】 プロセッサ及びコンパイラの提供側は、対象となるプロセッサに応じてintrinsics関数をコンパイラ内部に予め組み込んでおくことにより、プログラム作成者がプロセッサ特有の命令動作を高級言語にて記述することを可能にし、機械語にコンパイルすることを可能にしている。

【0007】従来、`intrinsics`関数はコンパイラ内部に組み込み済みであるため、ユーザが独自に`intrinsics`関数を定義することができなかった。仕様（プロセッサ・アーキテクチャおよび命令セット）が変化しないプロセッサ用のコンパイラにおいては、この方法で問題ない。

【0008】

【発明が解決しようとする課題】しかしながら、ユーザによる仕様（プロセッサ・アーキテクチャおよび命令セット）拡張可能なプロセッサ用のコンパイラでは、ユーザの仕様（プロセッサ・アーキテクチャおよび命令セット）拡張に応じた専用のコンパイラが必要となることから、ユーザが独自に`intrinsics`関数をカスタマイズできる必要がある。

【0009】そこで、本発明は、`intrinsics`関数をコンパイラの起動時および動作中に任意に追加定義することを可能にするコンパイラ、コンパイル方法及びコンパイルプログラムを記録したコンピュータ読み取り可能な記録媒体を提供することを目的とする。

【0010】

【課題を解決するための手段】上記課題を解決するため、本発明の特徴は、ソースプログラム中あるいはヘッダファイル中の組込関数の定義及び属性を読み取り、前記組込関数の定義及び属性に基づいて前記ソースプログラムを機械語に変換することにある。

【0011】また、本発明の他の特徴は、ソースプログラムを入力としオブジェクトコードを生成するコンパイラであって、前記ソースプログラム内に記述された命令をトークンに分割する字句解析部と、前記トークンが文法に適しているかを解析し、これらトークンの組み合わせが組込関数及び属性を定義しているかを解析する構文解析部と、前記構文解析部で解析された組込関数の定義及び属性を記憶する組込関数定義属性記憶手段と、前記組込関数の定義及び属性に基づいて、前記ソースプログラムを機械語へ変換するコード生成部とを有することにある。

【0012】また、組込関数の定義には、仮引数型と識別子名が含まれる。

【0013】本発明の特徴によれば、ソースプログラム中の組込関数（`intrinsics`関数）の定義と属性を解析して、それを記憶手段に記憶することによって、コンパイラの起動時及び動作時に`intrinsics`関数を任意に追加定義可能となる。

【0014】

【発明の実施の形態】以下、図面に基づいて本発明の実施の形態を説明する。以下の図面の記載において、同一又は類似の部分には同一又は類似の符号が付してある。

【0015】（1）第1実施形態

本実施形態では、`mov Rn, Rm`（機能：レジスタRmの内容をレジスタRnに代入する。コードサイズ：2バイト）という命令に関して`intrinsics`関数を定義する。以下、定

義方法、使用方法及びコンパイラでの実装方法に分けて記述する。

【0016】（1-1）`intrinsics`関数の定義方法

独自の予約語（`_asm, _reg_dest, _reg_src`）と`#pragma custom`を使用して、`intrinsics`関数を定義する。`mov Rn, Rm`の定義は、以下のように記述される。

【0017】`void_asm mov (int_reg_dest, int_reg_src)`；この定義方法は、`_asm`という独自キーワードを追加した以外には、ISO/JIS C言語規格（ISO/IEC 9899:1990）の関数宣言に従っている。この`_asm`を付加することにより、通常の関数宣言ではなく、`intrinsics`関数の定義であることを示している。

関数名`mov`により、`mov`命令の`intrinsics`関数の定義であることを示す。`_reg_dest, _reg_src`は、仮引数の識別子名であるので、ISO/JIS C言語規格上は、任意の識別子名が記述できる。仮引数は関数の定義で使われ、それが呼び出されるときに実引数で置き換えられる。`intrinsics`関数を定義する場合は、これらの特別な予約語を用いることにより、`mov`命令の動作についての情報をコンパイラに伝達する。`_reg_dest`はレジスタのデスティネーションのオペランド、`_reg_src`はレジスタのソースのオペランドであることを示す。本実施形態では、`_asm, _reg_dest, _reg_src`を予約語として使用したが、ISO/JIS C言語規格の予約語以外の語句であれば、他の語句を`intrinsics`関数定義用予約語として使用することができる。

【0018】次に、`#pragma custom`は、`mov`命令の場合は、以下のようになる。

【0019】`#pragma custom mov 2byte`

`#pragma`は、ISO/JIS C言語規格に規定されている前処理指令のひとつであり、処理系によって独自の拡張が許されている。本実施形態では、`#pragma custom`によって、`intrinsics`関数の色々な属性の定義を行う。この記述では、`mov`命令の命令長が2バイトであることを定義している。本実施形態では、`custom`という語句を使用したが、これは任意の語句で置き換え可能である。

【0020】（1-2）`intrinsics`関数の使用方法

下記のC言語プログラムのように、`intrinsics`関数を利用する前に、`intrinsics`関数の定義を行う。そして、通常の関数呼び出しの記述方法で、`intrinsics`関数を利用する。

【0021】<C言語プログラム>

```
/*intrinsics関数の定義*/
void_asm mov(int_reg_dest, int_reg_src);
#pragma custom mov 2byte
void test() {
    int a, b;
    /*intrinsics関数の利用*/
    mov(a, b);
}
```

コンパイラは、予め定義されたintrinsic関数の定義内容に従って、コンパイルを行い、以下のようなアセンブラファイルを生成する。指定されたとおり、`mov`命令が生成されている。

【0022】<生成されたアセンブラファイル>
`_test:`
`mov $1, $0`
`ret`

(1-3) intrinsic関数の実装方法

図1及び図2にコンパイラの処理の流れを示す。コンパイラは、高級言語で記述されたソースプログラムを入力とし、オブジェクトコードを出力とするソフトウェアプログラムである。

【0023】図1は中間コードを生成するコンパイラの構成を示す図である。図1に示すように、コンパイラ10aは、まず字句解析部11による字句解析を行う。

「字句解析」では、ソースプログラム1に記述された命令文を意味のある最小の単位であるトークンに分割する処理を行う。

【0024】次に、構文解析部12による構文解析を行う。「構文解析」では、直前の字句解析によって変数や記号など意味のある最小単位であるトークンに分割された命令文が、それぞれの言語で定められた文法に適しているかどうかをチェックする。

【0025】そして、中間コード生成部13による中間コードの生成、中間コード最適化部14による中間コードの最適化を経て、コード生成部15aによるコード生成を行う。「コード生成」では、それまでに行われたソースプログラム1の最小単位への分割、構文エラーのチェックなどの結果を受けて、コード・ジェネレータの機能を用いてソースプログラム1を、アセンブラ形式や2進数形式などの機械語へ変換する処理が行われる。

【0026】さらに、コード最適化部16aによるコード最適化を行う。「コード最適化」では、直前のコード生成部により変換された機械語に、実際の処理効率を向上させるための変更が加えられる。

【0027】そして、コード出力部17aからオブジェクトコード3を生成する。

【0028】コンパイラ10aが中間コード生成を行っているのは、構文解析の直後にコード生成をすると、生成されるプログラム・サイズが非常に大きくなり、変換処理が効率よく行われない場合があるため、ソースプログラム1aと等価で、しかも簡潔な言語である中間コードに変換してから翻訳処理を行うからである。中間コード生成及び中間コード最適化を省略してもオブジェクトコードは得られる。

【0029】図2は中間コードを生成しないコンパイラの構成を示す図である。図2に示すように、コンパイラ10bは、字句解析部11、構文解析部12、コード生成部15b、コード最適化部16b、コード出力部17

50

bを経て、ソースプログラム1aからオブジェクトコード3を生成する。

【0030】図1又は図2に示すいずれのコンパイラであっても、「構文解析」において、intrinsic関数の定義、使用を識別する必要がある。このための処理の流れを図3に示す。また、属性情報処理の流れを図4に示す。

【0031】図3は、intrinsic関数の定義の構文解析処理の流れを示す図である。図3に示すように、まず"_asm"が付加されているかをチェックする(ステップS30)。"_asm"が付加されていなければ、通常の関数の宣言であると認識する(ステップS31)。"_asm"が付加されている場合は、intrinsic関数の定義の処理をする(ステップS32)。まず、仮引数の型情報、識別子の名前の解釈を行う(ステップS33)。仮引数の型の指定方法等に間違いがある場合は、エラーメッセージを出力する(ステップS34)。仮引数の型の指定方法等に間違いが無い場合は、intrinsic関数の定義内容を記号表へ登録する(ステップS35)。記号表とは、定義したintrinsic関数やその引数をサーチするために使用する表である。

【0032】図4は、属性情報処理の流れを示す図である。図4に示すように、"#pragma"の次の字句が"custom"かチェックする(ステップS40)。"custom"でない場合は、他の#pragma指令の処理を行う(ステップS41)。"custom"である場合は、"#pragma custom"の処理を行う(ステップS42)。指定された識別子(例えば、`mov`)が、intrinsic関数として記号表に登録されているかチェックする(ステップS43)。記号表に登録されていない場合は、エラーメッセージを出力する(ステップS44)。記号表に登録されている場合は、指定された属性情報の解釈を行う(ステップS45)。指定された属性(字句)が解釈できない場合は、エラーメッセージを出力する(ステップS46)。指定された属性(字句、例えば2byte)が解釈できる場合は、intrinsic関数の情報に属性情報を付加する(ステップS47)。

【0033】本第1実施形態によれば、"_asm"が付加されているとintrinsic関数の定義の処理が開始され、仮引数の型情報や識別子名に誤りがなければ、intrinsic関数の情報として記号表に登録される。また、"#pragma"の次の字句が"custom"であれば、"#pragma custom"の処理が開始され、指定された識別子が記号表に登録されていれば、指定された情報(例えば、2byte)が属性情報として登録される。これにより、intrinsic関数がコンパイラの起動時及び動作中に定義可能となる。

【0034】図5に、intrinsic関数を使用した場合のコンパイラの処理の流れを示す。図5に示すように、関数コールの処理が開始されると、まず使用している関数名を記号表から探す(ステップS50)。そして、記号

表に登録されているか、`intrinsics`関数かをチェックする（ステップS51）。記号表に登録されているが、`intrinsics`関数ではない場合は、通常の関数の処理を行う（ステップS52）。記号表に登録されている`intrinsics`関数である場合は、指定された引数（実引数）と`intrinsics`関数宣言時の引数（仮引数）の情報をチェックする（ステップS53）。さらに、実引数と仮引数で、数及び型が一致するかをチェックする（ステップS54）。数又は型が一致しない場合は、エラーメッセージを出力する（ステップS55）。数及び型が一致する場合は、`intrinsics`関数としてコード（中間コード又はアセンブラーコード）を生成する（ステップS56）。

【0035】これによってユーザが定義した`intrinsics`関数を用いてコンパイルすることが可能となる。

【0036】（2）第2実施形態

第1実施形態では`mov`命令の実装方法を説明したが、第2実施形態では汎用的に使用する場合について述べる。

【0037】（2-1-1）仮引数の識別子に用いる予約語

第1実施形態で示した`_reg_src`と`_reg_dest`以外に、`_reg_modify`、`_mem_read`、`_mem_write`、`_mem_modify`、`_imm`、`_label`を独自の予約語として追加することにより、代表的な機械命令は`intrinsics`関数として記述可能となる。

【0038】`_reg_modify`は、ソースであり、かつデスティネーションでもあるレジスタのオペランドを示す。`_mem_read`はメモリ・リードのオペランドを示す。`_mem_write`はメモリ・ライトのオペランドを示す。`_mem_modify`はメモリ・リードと演算結果のメモリ・ライトのオペランドを示す。`_imm`は即値のオペランドを示す。`_label`は分岐先のラベル名を指定するオペランドを示す。

【0039】（2-1-2）`#pragma custom`で指定できる属性

第1実施形態で示した`mov`の属性（2byte）以外に、`"read"`、`"write"`、`"modify"`、`"interlock"`、`"freeze"`、`"uncondition"`、`"call"`、`"return"`の属性を定義することにより、代表的な機械命令が記述可能となる。`"read"`は`intrinsics`関数の定義で指定したオペランド以外に、ソースとして使用するオペランドを指定する。`"write"`は`intrinsics`関数の定義で指定したオペランド以外に、デスティネーションとして使用するオペランドを指定する。`"modify"`は`intrinsics`関数の定義で指定したオペランド以外に、ソースとデスティネーションとして使用するオペランドを指定する。

【0040】たとえば、東芝マイクロプロセッサTX39の`mul`命令の仕様では、`mul`命令は指定したオペランド以外に`low`レジスタ（\$lo）をデスティネーションとする。この場合には、`"#pragma custom mul write $lo"`と定義する。

【0041】`"read"`、`"write"`、`"modify"`を指定することにより、`intrinsics`関数が使用する資源の全ての情報を設定することができ、コンパイラによる機械命令の生成が可能となる。

【0042】`"interlock"`は、コンパイラに命令スケジューリングの必要性を指示する。連続する命令（例えば、命令Aと命令B）が、共通のオペランドを使用するためにパイプラインがストールしてしまう場合に、ストールの原因となっているオペランドを使用しない命令（例えば、命令C）を、共通のオペランドを使用する連続する命令（命令Aと命令B）の間に挿入することにより、ストールを防止する。

【0043】`"freeze"`は、この`intrinsics`関数の前後の命令の順番を変更しないことをコンパイラに指示する。

【0044】オペランドとして`_label`を指定した場合、その命令が条件分岐命令か、無条件分岐命令か判別できない。このため、条件分岐命令を既定値（default）とし、無条件分岐命令を指定する場合には`"uncondition"`を属性として指定する。無条件分岐命令を既定値（default）とし、条件分岐命令を指定する場合には`"condition"`を属性として指定する。

【0045】`"return"`は、関数コールからの復帰命令を`intrinsics`関数として定義した場合に使用する。

【0046】（2-2）次に、コプロセッサ命令を定義する方法を説明する。C言語プログラムは以下のように記述する。

【0047】

```
void_asm cpmov (int _cop_dest, int _cop_src);
#pragma custom cpmov cop 2byte
```

コプロセッサ命令であることを示すために、`#pragma custom`の属性に`"cop"`を追加する。これは、指定した関数（例えば、`"cpmov"`）が、コプロセッサ命令であることを示す。

【0048】また、仮引数の識別子に用いる予約語として、`_cop_src`、`_cop_dest`、`_cop_modify`を追加する。`_cop_src`は、コプロセッサ・レジスタのソース・オペランドであることを示す。`_cop_dest`はコプロセッサ・レジスタのデスティネーション・オペランドであることを示す。`_cop_modify`はコプロセッサ・レジスタのソース・オペランドでありかつデスティネーション・オペランドであることを示す。

【0049】（2-3）次に、VL IW（Very Long Instruction Word）やスーパースカラ型のプロセッサのコンパイラのために、`intrinsics`関数に対してスケジューリング属性を設定する方法を説明する。VL IWやスーパースカラでは、同時に演算できる命令に制限がある。このため、各命令に対して、どの演算器で実行できるかを設定する必要がある。この情報を`#pragma custom`の属性として設定することにより、VL IWやスーパースカラ型のプロセッサに対応することができる。

【0050】(2-4) また、intrinsics関数の第1引数が、"_reg_dest"または"_cop_dest"の場合は、関数の返却値として、設定することができる。このとき第2引数以降が存在する場合は、その第2引数以降は一つずつ繰り上がる。例えば、void _asm mov(int _reg_dest, int _reg_src);は、

```
int _asm mov(int _reg_src);
```

と同様の内容である。この場合、代入先を "=" 演算子で指定することができるようになり、C言語らしくなるため、可読性が向上する。

【0051】第1実施形態と同様の内容のプログラムが、以下のように記述される。

【0052】

```
int _asm mov(int _reg_src);
#pragma custom mov 2byte
void test0{
    int a, b;
    a = mov(b);
}
```

本第2実施形態によれば、コプロセッサ、VLIW又は 20 スーパースカラ型プロセッサのintrinsics関数がコンパイラの起動時及び動作中に定義可能となる。

【0053】また、代入先を "=" 演算子で指定することができるようになり、C言語らしくなるため、可読性が*

```
#ifdef __USE_MOV__ /* マクロ __USE_MOV__ が定義された場合の処理 */
/* intrinsics関数 */
int _asm mov(int _reg_src);
#pragma custom mov 2byte
#else /* マクロ __USE_MOV__ が定義されない場合の処理 */
/* 通常の関数 */
int mov(int src); /* この関数は、mov命令と同様の処理を行う */
#endif
/* 実際の使用 */
void test0{
    int i, j;
    /* intrinsics関数がどうかに関係なく使用できる */
    i = mov(j);
}
```

(3) 第3実施形態

第1、2実施形態においては#pragma customも用いて定義したが、第3実施形態では#pragma customを用いないで定義する方法を説明する。

【0058】#pragma customで指定した属性を関数の定義中に含める場合は、さらに独自の予約語を追加することになる。この場合、複数の予約語を追加する方法と1つの予約語を追加する方法とがある。

【0059】(3-1) 複数の予約語を追加する場合属性毎に、予約語(_2byte,_read,_write,_modify,_interlock,_freeze,_uncondition,_call,_return等)を追加することにより、#pragma customがなくても属性を設定 50

*向上する。

【0054】さらに、(1) ある命令(例えば、mov命令)が、「通常の関数」として実装されている場合と、機械命令として実装されているので「intrinsics関数」として定義される場合、の両方に対応するために、「intrinsics関数の引数と返り値の形式」を「通常の関数の引数と返り値の形式」と同じにする、(2) コンパイラが持っている条件コンパイルを利用し、起動時に指定するコマンドライン・オプションによって、「intrinsics関数」として定義するか、「通常の関数」として定義するかを選択する、ことにより、ソースプログラムを記述する際に、ある命令(例えば、mov命令)が「通常の関数」として実装されているか、「intrinsics関数」として定義されているかを、プログラマーが意識する必要がなくなる。このため、プログラミングが容易となり、作成したプログラムの再利用性も向上する。

【0055】なお、条件コンパイルとは、Cコンパイラでは、#ifdef, #else, #endif として実装されており、コンパイラ起動時に指定するマクロ定義のコマンドライン・オプションによって、コンパイルする内容を変更することができる機能のことである。

【0056】条件コンパイルの例を以下に示す。

【0057】

することが可能となる。例えば、第1実施形態のmov命令は、void _asm_2byte mov(int _reg_dest, int _reg_src);と定義することができる。

【0060】(3-2) 1つの予約語を追加する場合属性毎に予約語を追加するのではなく、1つの予約語を追加することによって、定義することもできる。"_attr"という予約語を定義した場合、第1実施形態のmov命令は、void _asm_attr(2byte) mov(int _reg_dest, int _reg_src);と定義することができる。この場合、"_attr"の構文分析時に、指定された属性をチェックする。予約語が少ない方が、プログラミング上の制約が少なくなるため好ましい。

11

【0061】なお、上記第1～3実施形態ではintrinsic関数に必要な情報を、コンパイルするソースファイル中に記述したが、別ファイルからintrinsic関数に必要な情報を読み込むことも可能である。図6に、intrinsic関数の情報を別ファイルとする場合の処理の流れの概略を示す。図6に示すように、ソースファイル1bとintrinsic関数情報ファイル60をコンパイラ10に入力し、オブジェクトコード3を得る。

【0062】この場合は、第1～3実施形態で示したような予約語などに拘束されることはないので、intrinsic関数に必要な情報を任意の書式で記述することが可能となる。

【0063】

【発明の効果】上記の如く、本発明によれば、intrinsic関数の動作に必要な情報をコンパイラの外部から設定できるようにすることにより、コンパイラを容易に拡張することができる。このため、ユーザがコンパイラをカスタマイズすることもできる。

【0064】また、intrinsic関数の定義を、コンパイルの対象であるソースファイル内に混在させることにより、以下の効果も得られる。

【0065】1) intrinsic関数の定義は、通常の関数定義にいくつかの予約語を追加しただけなので、ユーザが容易に理解することができる。

【0066】2) ユーザに、C言語のヘッダファイルと*

10

20

20

*して提供できるため、リリースが容易になる。

【0067】3) intrinsic関数の第1引数が、ディスティネーションの場合は、関数定義を変更することにより、ソースファイル中のintrinsic関数名（例えば、mov関数）と同じ名称の命令（例えば、mov命令）を使用する部分を変更しなくてすむ。これにより、プログラムの再利用性が向上する。

【図面の簡単な説明】

【図1】中間コードを生成するコンパイラの構成を示す図である。

【図2】中間コードを生成しないコンパイラの構成を示す図である。

【図3】intrinsic関数の定義処理の流れを示す図である。

【図4】属性情報処理の流れを示す図である。

【図5】intrinsic関数を使用した場合のコンパイラの処理の流れを示す図である。

【図6】intrinsic関数の情報を別ファイルとする場合の処理の流れの概略を示す図である。

【符号の説明】

1 ソースプログラム

3 オブジェクトコード

10 コンパイラ

11 字句解析部

12 構文解析部

13 中間コード生成部

14 中間コード最適化部

15a コード生成部

16a コード最適化部

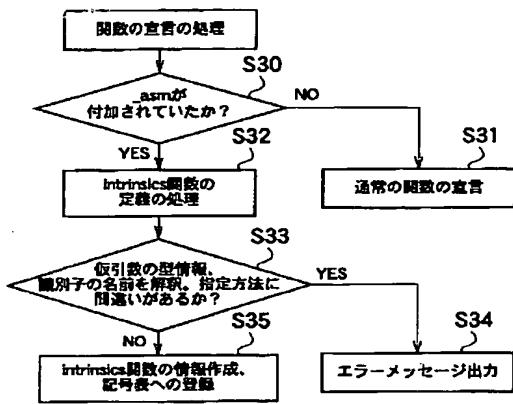
17a コード出力部

3 オブジェクトコード

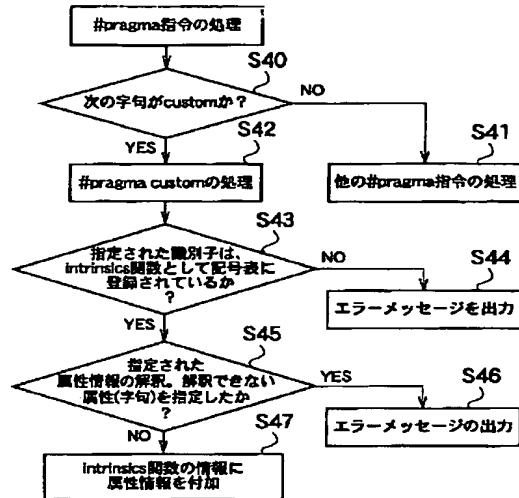
【図1】

【図6】

【図3】



【図4】



【図5】

